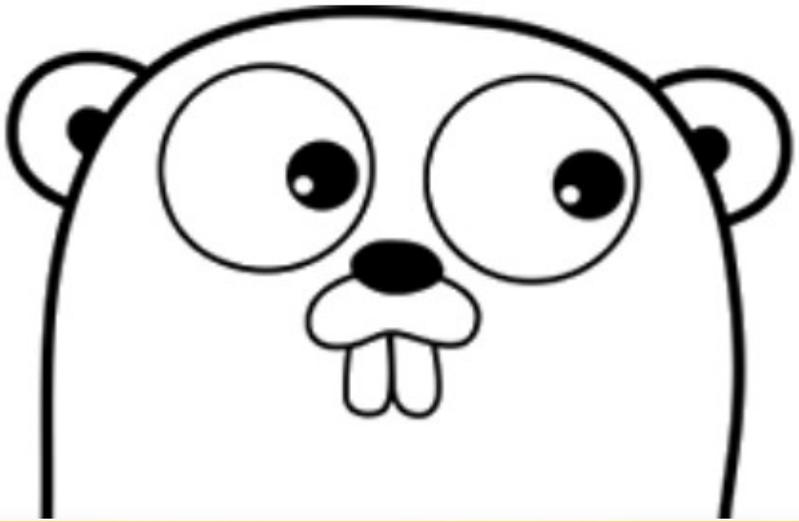
An introduction to Go

Kevin Golding

What is Go?

Go is an open source programming language that makes it easy to build **simple**, **reliable**, and **efficient** software.



Hello World!

```go
package main

import (
    "fmt"
)

func main() {
    // English
    fmt.Println("Hello world")

    // Chinese - because go source code is UTF-8!
    fmt.Println("你好，世界")
}
```

```
$ go run 01-HelloWorld.go
Hello world
你好，世界
```

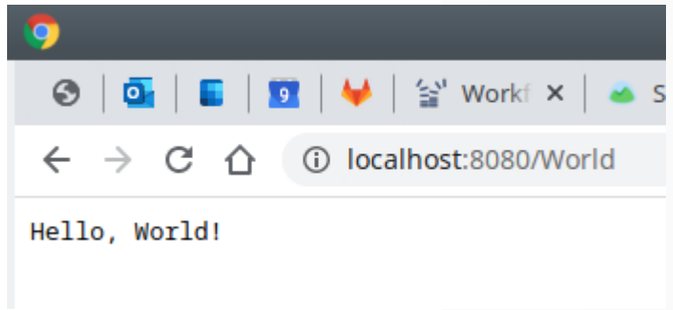
Hello World (Web server version)

```go
package main

import (
    "fmt"
    "net/http"
)

func main() {
    http.HandleFunc("/", HelloServer)
    http.ListenAndServe(":8080", nil)
}

func HelloServer(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello %s!", r.URL.Path[1:])
}
```



02-HelloWebServer.go

Hello World (Desktop application version)

```go
package main

import (
    "fyne.io/fyne"
    "fyne.io/fyne/app"
    "fyne.io/fyne/widget"
)

func main() {
    a := app.New()

    w := a.NewWindow("Hello World")
    w.SetContent(widget.NewVBox(
        widget.NewLabel("Hello World!"),
        widget.NewButton("Quit", func() {
            a.Quit()
        }),
    ))
    w.Resize(fyne.NewSize(400, 300))
    w.ShowAndRun()
}
```

03-HelloDesktopApp.go

Go Elevator pitch

- Concurrency – *simple, efficient & performant*
- Statically typed – *a number is a number not a string*
- Single file compiled binaries - *cross-platform*
- Powerful well documented standard libraries
- Built-in testing, profiling, package management
- Opinionated, clean, predicable looking code

Keywords

`break`	`default`	`func`	`interface`	`select`
`case`	`defer`	`go`	`map`	`struct`
`chan`	`else`	`goto`	`package`	`switch`
`const`	`fallthrough`	`if`	`range`	`type`
`continue`	`for`	`import`	`return`	`var`

Just 25 core language keywords!

Types

- Numeric
  - Unsigned integers
  - Integers
  - Floating point
  - Complex
- Aliases
  - Byte = Uint8
  - Rune = int32
- String
- Array
- Slice
- Bool
- Struct
- Pointer
- Function
- Interface
- Map
- Chan

Variables

- Typed (except Interface)
- Scoped global or local
- Automatic garbage collection
- Pointers
  - Address of: &name
  - Pointer: *name

- Variable declarations
  - var s string
  - var s string = “Hello”
  - s := “I’m a string”
  - i := 56
  - i := int(56)
  - b := []byte{0x1a, 0x ff}
  - m := map[string]int{
    ”key”: 999
    }

Constants

- Typed
- Scoped global or local
- Autocomplete (iota)

- Constant declarations
  - const s = “I AM A STRING”
  - const (
    MIN_AGE = 1
    MAX_AGE = 999
)
  - const (
    KEY0 = iota
    KEY1
    KEY2
)

Array & Slices

- Arrays are fixed length!
  - `var a [5]int`
  - `a := [5]{1, 2, 3, 4, 5}`
- Slices are references to arrays
  - Flexible wrapper on top of an array
  - `var s []int`
  - `s := []int{1, 2, 3, 4, 5}`
- Push x onto a
  - `a = append(a, x)`
- Pop x from a
  - `x = a[len(a)-1]`
  - `a = a[:len(a)-1]`
- Iteration
  -
```
for i, x := range a
{
    // x is a[i]
}
```

Code break...

- Golang playground
  - https://play.golang.org/
  - Alternative with syntax highlighting: https://goplay.space/
- Install on local machine (Linux, Windows or OS X)
  - https://golang.org/
  - Free editors
    - Visual Studio Code with golang extensions
    - LiteIDE https://github.com/visualfc/liteide

Code break... iterate over an array

```go
package main

import (
    "fmt"
)

func main() {
    a := []string{"Zero", "One", "Two", "Three", "Four", "Five"}

    for i, s := range a {
        fmt.Println("Index", i, "is", s)
    }
}
```

04-Loop.go

Functions

- Can return multiple values
- Functions can be defined and passed just like a variable
  - Enables callbacks
- Parameters can be either
  - Passed by value
  - Passed by reference using pointers

```go
package main

import "fmt"

func swap(x, y string) (string, string) {
    return y, x
}

func main() {
    a, b := swap("world", "hello")
    fmt.Println(a, b)
}
```

07-Functions.go

Go routines

- “Goroutines are functions or methods that run concurrently with each other”
- Lightweight aka cheap
  - Goroutine !== OS Thread
- “Don't communicate by sharing memory, share memory by communicating”
  - Channels
  - Mutex’s

Code break... go routines

```go
package main

import (
    "fmt"
    "time"
)

func main() {
    a := []string{"Zero", "One", "Two", "Three", "Four", "Five"}

    for i, s := range a {
        go output(i, s)
    }
    time.Sleep(time.Millisecond)
}

func output(i int, s string) {
    fmt.Println("Index", i, "is", s)
}
```

05-GoLoop.go

Code break... go routines - sync

```go
package main

import (
    "fmt"
    "sync"
)

func main() {
    a := []string{"Zero", "One", "Two", "Three", "Four", "Five"}
    var wg sync.WaitGroup
    for i, s := range a {
        wg.Add(1)
        go output(i, s, &wg)
    }
    fmt.Println("Waiting...")
    wg.Wait()
    fmt.Println("All done")
}

func output(i int, s string, wg *sync.WaitGroup) {
    defer wg.Done()
    fmt.Println("Index", i, "is", s)
}
```

06-WaitGroup.go

Packages

- Aka libraries
- Built-in package manager, versioning & vendoring
- Standard library is built-in
  - http, fmt, errors, log, testing, bytes, net, os, regexp, sort…
- External packages use full path
  - e.g. import "fyne.io/fyne/app"
- “go get” will download latest imported packages

Struct

- Collection of fields
- Like an Object
- Can have methods
- Implied interfaces
  - If it behaves like X then it is an X
- Embedding

```go
package main

import "fmt"

type Person struct {
    Name string
    Age  int
}

func (p Person) String() string {
    return fmt.Sprintf("%v (%v years)", p.Name, p.Age)
}

func main() {
    a := Person{"Arthur Dent", 42}
    z := Person{"Zaphod Beeblebrox", 9001}
    fmt.Println(a, z)
}
```

08-Structs.go

Public/Private

- Public = Exported
  - First letter must be uppercase
- Private = Unexported
  - First letter must be lowercase
- Applies to packages and structs

Code break... JSON

```go
package main

import (
    "encoding/json"
    "net/http"
)

type Person struct {
    Name string `json:"name"`
    Age  int    `json:"age`
    note string `json:"note"`
}

func main() {
    http.HandleFunc("/", JsonServer)
    http.ListenAndServe(":8080", nil)
}
```

```go

func JsonServer(w http.ResponseWriter, r *http.Request) {
    a := Person{"Arthur Dent", 42, "HHGTG"}
    buf, err := json.Marshal(a)
    if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        w.Write([]byte(err.Error()))
        return
    }
    w.Header().Set("Content-Type", "application/json")
    w.Write(buf)
}
```

09-Json.go

Channels

- Channels are “pipes” that let you communicate safely between goroutines
  - **MyChannel := make(chan string)**
  - Sending: **MyChannel <- “Hello”**
  - Reading: **str := <- MyChannel**
- Reads & Writes can block
- Can be buffered using make(chan string, 100)

Code break... channels

```go
func main() {
    intChan := generator()

    fmt.Println("Waiting for first item")
    firstItem := <-intChan
    fmt.Println("First item is", firstItem)

    for item := range intChan {
        fmt.Println("Received item", item)
    }
    fmt.Println("intChan has closed")
}

```

```go
func generator() chan int {
    myChan := make(chan int)
    go func() {
        i := 0
        for {
            time.Sleep(time.Second)
            myChan <- i
            if i == 5 {
                close(myChan)
                return
            }
            i++
        }
    }()
    return myChan
}
```

10-Channels.go

Code break... more channels

```go
func main() {
    intChan := generator()

    timeOut := time.After(time.Second * 6)

    for {
        select {
        case item, ok := <-intChan:
            if !ok {
                fmt.Println("generator finished")
                return
            }
            fmt.Println("Received item", item)

        case <-timeOut:
            fmt.Println("timed out")
            return
        }
    }
}
```

```go
func generator() chan int {
    myChan := make(chan int)
    go func() {
        i := 0
        for {
            time.Sleep(time.Second)
            myChan <- i
            if i == 5 {
                close(myChan)
                return
            }
            i++
        }
    }()
    return myChan
}
```

11-Channels.go

Resources

- Go main site (excellent tutorials)
  - golang.org
- Editors
  - github.com/visualfc/liteide
  - code.visualstudio.com
- Go Playgrounds
  - play.golang.org
  - goplay.space
- Tutorials
  - gobyexample.com
  - github.com/golang/go/wiki
- Slides and sample code
  - github.com/kgolding/present-go-intro

Coding challenge

Level 1 challenge

- Extend the network chat server to respond to a new command
  - “/time” and respond with a timestamp

Level 2 challenge *(no copying from the server source)*

- Extend the network chat client to respond to commands:
  - “?time” and respond with a timestamp
  - “?uptime” and respond with a duration

Level 3 challenge

- Create a desktop application network chat client!

WIFI: RedSprite_2G / redsprite
github.com/kgolding/present-go-intro
./code/go-chat-server.go
./code/go-chat-client.go

The end slide of an introduction to Go

Kevin Golding